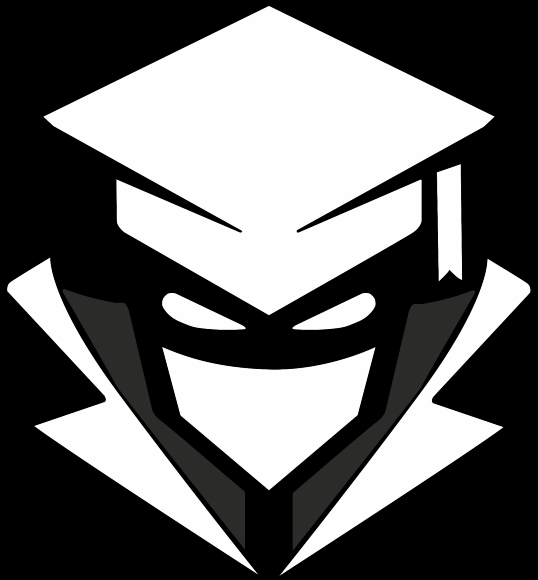


MISC

PROTOCOLS & THE WEB

IF YOU HAVEN'T ALREADY

- **Sign up:** www.umisc.info/signup
- If you're on Windows, consider installing WSL https://aka.ms/wslinstall
- GitHub for slides: https://github.com/umisc/workshops

WORKSHOP OVERVIEW

- OSI Model & TCP/IP Model
- The Internet
- HTTP
- Tools
- Other Protocols
- Solve some challenges!

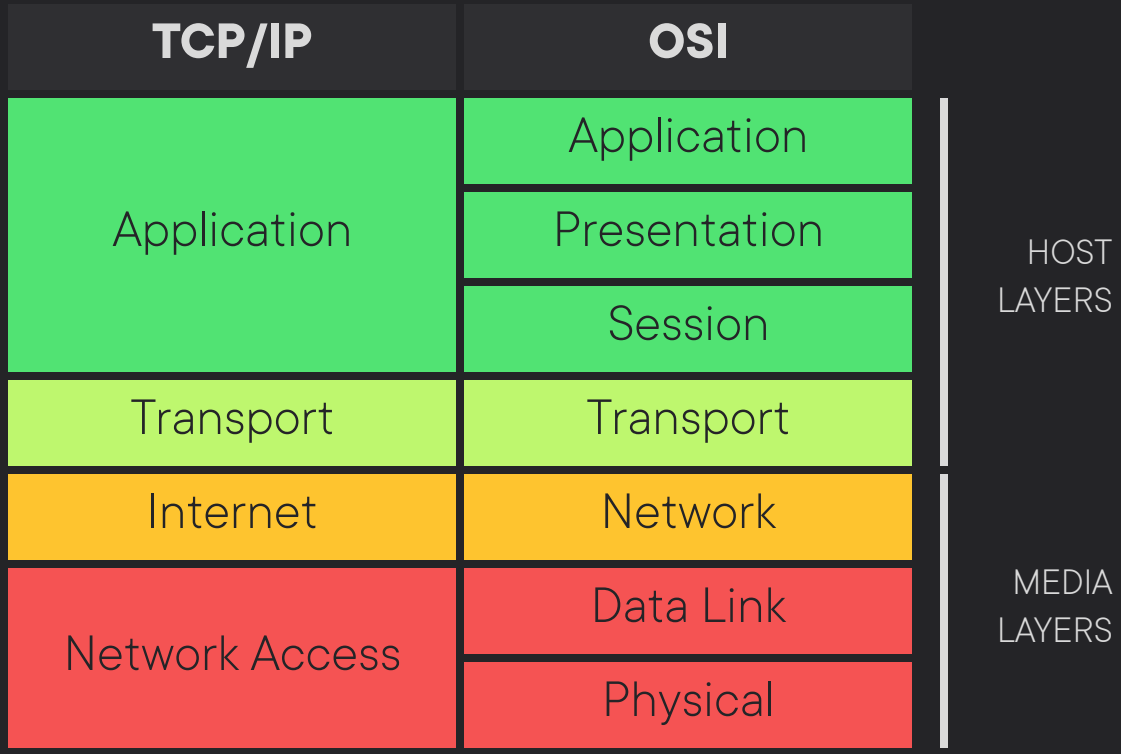
OSI MODEL

- Conceptual model to **describe a method** for communications between computers
- Allows for interoperability with **standard protocols**
- Consists of **7 layers**, each perform a specific function and interact only with **adjacent layers**
- Models communication all the way from raw electrical signals to high level applications

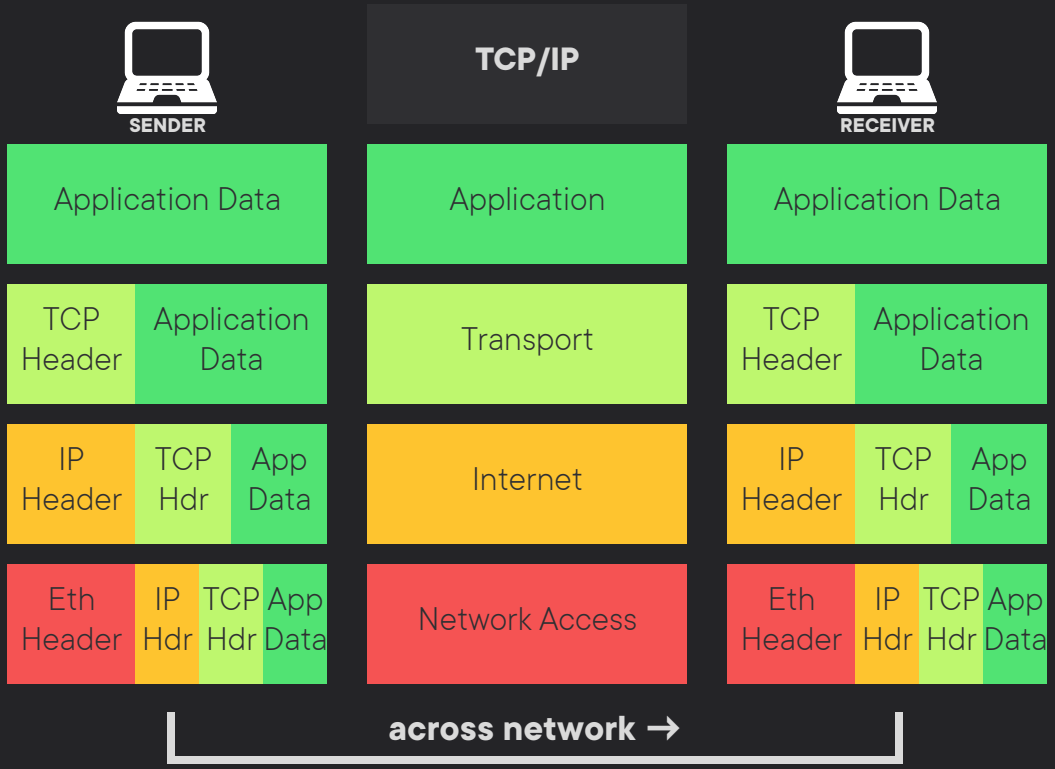
TCP/IP MODEL

- Conceptual model similar to the OSI model
- More **pragmatic** than OSI
  - Has become the **standard** for networking (and the internet)
- Consists of **4 layers** which can be *roughly mapped* to the 7 layers of the OSI model

OSI VS. TCP/IP


TCP/IP
OSI
Application
Application
Presentation
Session
Transport
Transport
Internet
Network
Network Access
Data Link
Physical
HOST LAYERS
MEDIA LAYERS

{EN,DE}CAPSULATION



THE INTERNET

- A massive **network of networks**
- Follows the TCP/IP model
- Every host has an **IP address**
- Programs/services **listen** on a port on a host
- Devices external to the network can connect to a service using a **public IP address** and a **port**

APPLICATION PROTOCOLS

WHAT IS A PROTOCOL?

- A set of rules that **describe** how to communicate
- Most protocols have public documents (**RFCs**) which act as **specifications** and **implementation guides**
- Once there is a TCP connection established, application protocols are used to allow for **meaningful** conversations

HTTP IS THE PROTOCOL THAT RUNS THE WEB

HTTP

- Specifications in RFC 2616
- Client/server model:
  - Client **requests** data and resources
  - Server **responds** with data and resources
- **Stateless**
- Used to transfer HTML documents, CSS (stylesheets), JS (scripts), images, videos, etc.
- A HTTP conversation consists of **request** and **response** messages

AN ASIDE ON URLS

http://www.google.com/search?q=hello

scheme | hostname | path | query

HTTP REQUESTS

A HTTP request message consists of:

- A request line
  - e.g. `GET /search?q=hello HTTP/1.1`
  - generic form `<METHOD> <PATH> <VERSION>`
- Request header fields (can be many lines)
  - e.g. `Host: google.com`
  - generic form: `Header: Value`
- An empty line
- An optional message body

THAT'S IT

HTTP RESPONSES

A HTTP response message consists of:

- A status line
  - e.g. `HTTP/1.1 200 OK`
  - generic form `<VERSION> <STATUSCODE> <MSG>`
- Response header fields (can be many lines)
  - e.g. `Content-Type: text/javascript`
  - generic form: `Header: Value`
- An empty line
- An optional message body

HTTP REQUEST METHODS

Method	Description
`GET`	Used to retrieve data; should have no side effects
`HEAD`	Requests for a response similar to a `GET` request, but includes only the response headers
`POST`	(Usually) sent with a body message and often causes a side effect on the server (e.g. saving something in a database)
`PUT`	Creates/updates a resource. Similar to a `POST` request, except multiple `PUT` request with the same body message has the same effect as just one
`DELETE`	Deletes the specified resource
`OPTIONS`	Requests for the capabilities of the server without implying an action on the resource

and more...

HTTP REQUEST HEADERS

Name	Description
`Connection`	Controls the connection (`close`, `keep-alive`, `...`)
`Content-Length`	Length of the request body (for `POST` and `PUT` requests)
`Content-Type`	Type of data being sent in the request body (for `POST` and `PUT` requests)
`Cookie`	To maintain state (more later...)
`Host`	Specifies the domain name of the server. Mandatory (to enable response to different pages on the same server)
`Referer`	Address of the previous web page
`User-Agent`	Identifies the user agent (client) making the request

and many more...

HTTP RESPONSE HEADERS

Name	Description
`Connection`	Same as the request `Connection` header
`Content-Length`	Length of the response body
`Content-Type`	Type of data in the response body
`Location`	Used for redirects
`Server`	Names the server responding to the request
`Set-Cookie`	Instructs the client to set a cookie. The server can send multiple `Set-Cookie` headers to set multiple cookies

and many more...

HTTP RESPONSE CODES

Code	Status Message	Description
`200`	`OK`	Standard response for successful requests
`301`	`Moved Permanently`	Redirect to the URL given in the `Location` header
`400`	`Bad Request`	Client side error (e.g. invalid request body, request body too large, etc.)
`401`	`Unauthorized`	Authorization required to access resource
`403`	`Forbidden`	User has insufficient permissions for requested resource
`405`	`Method Not Allowed`	Unsupported request method for the requested resource
`500`	`Internal Server Error`	Generic error message server-side errors

and many more...

HTTP EXAMPLE #1

Request:

```
HEAD / HTTP/1.1
Host: www.umisc.info
```

Response:

```
HTTP/1.1 301 Moved Permanently
Date: Fri, 06 Mar 2020 13:00:06 GMT
Connection: keep-alive
Cache-Control: max-age=3600
Expires: Fri, 06 Mar 2020 14:00:06 GMT
Location: https://www.umisc.info/
Server: cloudflare
CF-RAY: 56fc41e48bdedf85-MEL
```

HTTP EXAMPLE #2

Request:

```
POST /login HTTP/1.1
Host: www.example.com
Content-Type: application/json
Content-Length: 48

{"username":"misccat","password":"miscpassword"}
```

Response:

```
HTTP/1.1 200 OK
Server: Express
Set-Cookie: session_id=68616820676f6f64206f6e652121
```

HTTP EXAMPLE #3

Request:

```
GET /search?q=hello HTTP/1.1
Host: google.com
```

Response:

```
HTTP/1.1 200 OK
Server: gws
Content-Type: text/html
...

<!doctype html><html lang="en-AU"><head><meta charset="UTF
-8"><meta content="/images/branding/googleg/1x/googleg_sta
ndard_color_128dp.png" itemprop="image"><title>hello - Goo
gle Search</title>...
```

COOKIES

- Cookies help to **maintain state**
- Servers instruct clients to store a cookie with the `Set-Cookie` response header.
- Client's responsibility to keep and send the cookie in the `Cookie` request header for future requests
- Almost all websites with a login feature uses cookies to **store sessions**
    - It can be very bad if the session isn't saved and validated **server-side**

HTTP EXAMPLE #4

Request 1:

```
GET /dashboard HTTP/1.1
Host: www.example.com
User-Agent: my hands
```

Response 1:

```
HTTP/1.1 302 Found
Server: nginx
Location: http://www.example.com/login
Connection: keep-alive
```

HTTP EXAMPLE #4 ...

Request 2:

```
GET /login HTTP/1.1
Host: www.example.com
User-Agent: my hands
```

Response 2:

```
HTTP/1.1 200 OK
Server: nginx
Connection: keep-alive
Content-Type: text/html

<html><head><title>Login Page</title></head><body><form ac
tion="/login" method="post"><input type="text" name="usern
ame"><input type="text" name="password"><input type="submi
t" value="Submit"></form></body></html>
```

HTTP EXAMPLE #4 ...

Request 3:

```
POST /login HTTP/1.1
Host: www.example.com
User-Agent: my hands
Content-Type: application/x-www-form-urlencoded
Content-Length: 29

username=admin&password=admin
```

Response 3:

```
HTTP/1.1 302 Found
Server: nginx
Connection: keep-alive
Set-Cookie: session_id=646d56796553426a62323973495l6f3d
Location: http://www.example.com/dashboard
```

HTTP EXAMPLE #4 ...

Request 4:

```
GET /dashboard HTTP/1.1
Host: www.example.com
User-Agent: my hands
Cookie: session_id=646d56796553426a623239734951 6f3d
```

Response 4:

```
HTTP/1.1 200 OK
Server: nginx
Connection: keep-alive
Content-Type: text/html

((dashboard html goes here...))
```

HTTP EXAMPLE #5 😟

Request:

```
POST /login HTTP/1.1
Host: www.example.com
User-Agent: my hands

username=user&password=p4ssW0rd!
```

Response:

```
HTTP/1.1 302 Found
Server: nginx
Connection: keep-alive
Location: http://www.example.com/dashboard
Set-Cookie: username=user
Set-Cookie: admin=False
```

TAKEAWAYS

- HTTP requests and responses are just **pieces of text** formatted according to the specifications
- It's up to the server how to interpret *most* of the things sent in the request
  - For example, the server may decide to respond `403` to all requests with the `User-Agent` containing the word `"Windows"`
- Your web browser automatically generates and sends data, but that doesn't mean you can't edit it yourself!
- **Always analyse network requests!**

USEFUL TOOLS

- **nc** - create TCP and UDP connections, also has the ability to listen for TCP and UDP connections. Comes installed with most Linux distributions
- **curl** - transfer data using URLs. Comes installed with most Linux distributions
- **httpie** - easy to use command line HTTP client
- Burp Suite - GUI program with a proxy (amongst other things) to intercept and edit HTTP traffic
- Firefox - good web browser with good dev tools
- pwntools - python library that makes life easier

NC USAGE

- Connect to host `example.com` listening on port `443`:
  - `nc example.com 443`
- Connect to host `example.com` listening on port `443` and send `CRLF` line-ending (e.g. for SMTP and HTTP):
  - `nc -C example.com 443`
- Listen for TCP connections on port `1337`:
  - `nc -lvp 1337`

HTTPIE USAGE

- `GET` request to `http://google.com`:
  - `http google.com`
- `POST` JSON data to `example.com/login`:
  - `http POST example.com/login user=admin pass=admin`
- `GET` request to `example.com` with custom header:
  - `http example.com MyHeader:Value123`
- `GET` request to `example.com`, print headers:
  - `http -p hH example.com`

PWNTOOLS USAGE

Connect to host `google.com` listening on port `80` and send/receive some data:

```
from pwn import *
conn = remote('google.com', 80)
conn.sendline(b'GET /search?q=hello HTTP/1.1\r')
conn.sendline(b'Host: www.google.com\r')
conn.sendline('\r')
print(conn.recv().decode())
```

Results in:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=ISO-8859-1
...
```

SMTP

- Specified in RFC 5321
- Assigned to port 25
- Basic procedure as follows:
  - `HELO [domain_name]` identifies the client
  - `MAIL FROM:<[email_addr]>` initiates mail transaction
  - `RCPT TO:<[email_addr]>` identifies recipient(s)
  - `DATA` - begins reading data to be sent

FTP

- Specified in RFC 959
- Assigned to port 21
- Used to transfer files between a client and a server
- Can be configured to allow **anonymous access**
- Most browsers implement an FTP client

CHALLENGES!

http://workshop-ctf.umisc.info

- Easy: **Basics I**, **Basics II**, **Basics III**, **Hidden FTP**
- Medium: **Misccat's Flag Delivery Service**, **TCP Phonebook**
- Hard: **HTTP Server 3000**